

Gustavo Alonso

Swiss Federal Institute of Technology Zürich

Claus Hagen

Credit Suisse

Divyakant Agrawal and Amr El Abbadi

University of California, Santa Barbara

C. Mohan

IBM Almaden Research Center

Today's commercial workflow systems, although useful, do not scale well, have limited fault tolerance, and don't interoperate well with other workflow systems. The authors discuss current research directions and potential future extensions that might enable workflow services to meet the needs of mission-critical applications.

Enhancing the Fault Tolerance of Workflow Management Systems

The information resources of many modern corporations can best be characterized as a collection of widely heterogeneous, largely distributed, and loosely coupled computing environments. Many current trends reinforce this characterization: the decentralization of the corporation and of decision making, the emphasis on client-

server architectures, the relevance of federated systems, and the increasing availability of distributed-processing technology (the Web, CORBA, Object Linking and Embedding, Java), to mention a few. All these trends promote the deployment of large and heterogeneous distributed execution environments where interrelated tasks can be efficiently executed and closely monitored.¹

It is in this context that workflow tools are relevant.² Workflow management systems (WFMSs) are used to coordinate and streamline business processes.³ Typical examples include billing and loan approval in financial corporations, claim processing in insurance companies, and patient admittance in a health care organization. Workflow tools represent business processes as workflow processes—that is, as computerized models of the business process where all aspects necessary for executing the process are specified. These aspects include defining each individual step in the process,

the order and conditions in which the steps must be executed, the data flow between steps, identifying who is responsible for each step, and the applications to use at each step. A WFMS can thus be seen as the set of tools used to design and define workflow processes, the environment in which these processes are executed, and the corresponding set of interfaces to the users and applications.

The demand for workflow systems has grown so rapidly that in just a few years several hundred products have appeared in the market. In a recent issue of *IEEE Concurrency*, a special section on WFMSs discussed the bright perspectives of this technology, highlighting application areas such as enterprise-wide workflow management and interorganizational workflows.⁴ In spite of the overall success, however, there have been many cases of disappointing mismatches between users' expectations and the features provided. In most cases, the main reason for the problems has been

the immaturity of the first generation of workflow products. These problems are slowly being solved as products mature and enrich their functionality, but in a few areas, workflow systems are far from providing the functionality, reliability, and robustness necessary to become a key tool in corporate computing.

An overview of workflow systems

We will introduce the basic concepts of workflow management using the definitions provided by the Reference Model³ of the Workflow Management Coalition (WFMC), an international organization leading efforts to standardize workflow management products.

TYPES OF WORKFLOW SYSTEMS

In spite of the WFMC's efforts, people use the notion of workflow differently in different contexts. A widely used taxonomy of workflow systems is based on the nature of the business processes they are able to support, distinguishing between administrative, ad hoc, collaborative, and production processes.⁵ The basic parameters of this classification are the similarities among the business processes involved and their value to the associated enterprises. However, we can also organize them according to task complexity and structure. Figure 1 summarizes both approaches. Note, however, that there are no sharp distinctions and the different types do overlap. Furthermore, although many of the WFMC's definitions apply to all types of business processes and workflow systems, they are most appropriate for administrative and production workflows.

In general, *administrative* workflows refer to bureaucratic processes where the steps to follow are well-established and are based on known rules. Examples are registering for a university course, applying for a degree after finishing a dissertation, registering a vehicle, and almost any other process in which one must fill out and route a set of forms through a series of steps. This type of workflow leads almost naturally to the idea of *form processing*, a new term for the older con-

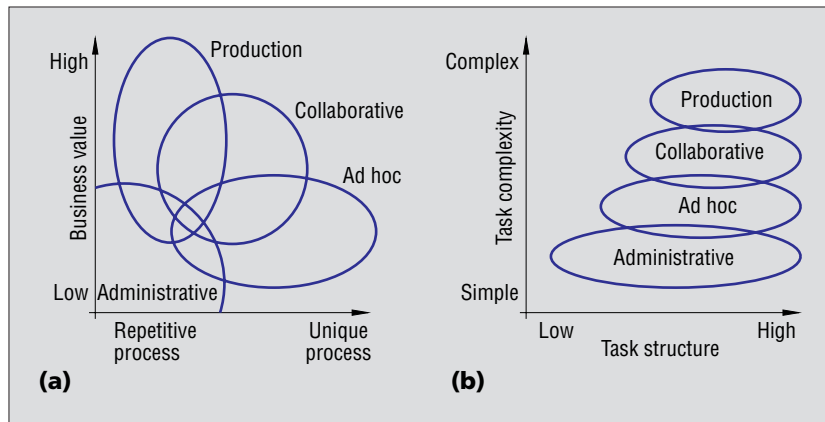


Figure 1. A rough classification of workflow management systems organized according to (a) their business value and uniqueness and (b) their task complexity and structure.

cept of the paperless office; it is also associated with large-scale systems where the number of processes involved tend to be very high. For instance, a typical billing application might involve several million process instances a year.

Ad hoc workflows address exceptions and unique situations. The situation might not be exceptional but each particular instance is unique. For example, each journal follows a different protocol for its submission process. Editors, especially given the length of time these processes run, might want to leave the coordination of the different steps in the hands of an ad hoc workflow system. Also, while the actual process might be unique, the user will generally be involved in a variety of processes. The reason for using a workflow system with these characteristics is not the difficulty of tracking each separate process but the problem of keeping track of them all simultaneously.

The third class, *collaborative* workflows, is mainly characterized by the number of participants involved and their interactions. Unlike other types of workflow that assume continual forward progress, a collaborative workflow might involve several iterations over the same step until the participants reach some agreement; it might even involve going back to an earlier stage. A good example is when several authors write a paper. It would be difficult to model such a process using tools that are not geared for collaboration, because it is almost impossible to predefine the steps to follow. (These steps should not be mistaken with milestones, which can be predefined.) Moreover, collaborative workflows tend to be dynamic in the

sense that they are defined as they progress. Taken to the extreme, this type of process might not even be a workflow system, because people do most of the coordination while the system just provides a good interface for recording their interactions.

Production workflows are the high end of these systems. We can characterize them as the implementation of mission-critical business processes—that is, those directly related to the organization's function. Credit and loan applications and insurance claims are typical examples. The difference between administrative and production workflows is sometimes a matter of perspective. Usually, when talking about production workflows, the main points to consider are the scale, the complexity and heterogeneity of the execution environment, the variety of people and organizations involved, and the nature of the tasks. In particular, production workflows tend to be executed over heterogeneous systems (frequently legacy applications), and monitoring tools with which to statistically analyze process execution are crucial. Most commercial tools focus on production workflows, because these have the biggest market potential; on the other hand, availability and robustness are most critical here.

WORKFLOW MODEL

At the core of any workflow system is the notion of a business process. The reference model defines a business process as "a procedure where documents, information, or tasks are passed between participants according to defined sets of rules to achieve, or contribute to, an overall business goal."³ A workflow is a representation of the business process in a

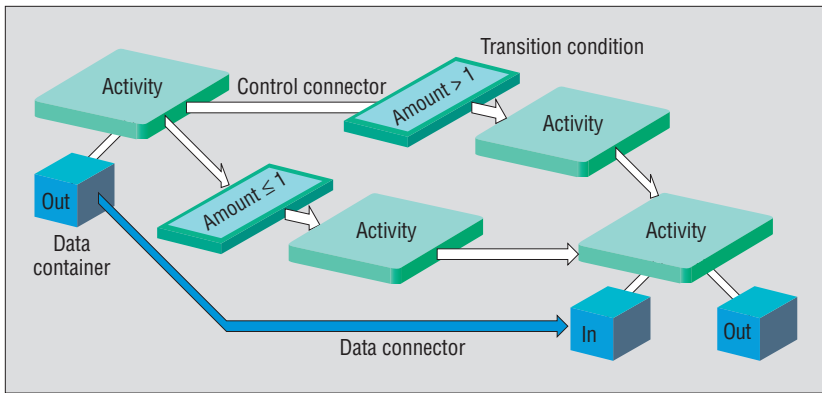


Figure 2. The main components of FlowMark's process model for control and data flow.

machine-readable format. Hence, a WFMS is “a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic.”³

Figure 2 shows the main components of a process, using IBM's FlowMark product as an example. It consists of a set of activities, a set of data objects, and a number of edges connecting the components. An activity is the internal representation of a program execution or manual task. An activity's connection to the “real world” is encoded in its external binding, which specifies which program to run, who is responsible for performing this part of the workflow, and which resources to allocate for its execution. The runtime system uses this information to execute external applications or instruct users to perform certain actions. The data objects are grouped in data containers that represent the input and output parameters of activities and are used to pass information around.

Control connectors—directed edges that link pairs of activities and specify their order of execution—determine control flow inside a process. To enable conditional branching, control connectors can be marked with transition conditions that are based on the state of data objects and are evaluated at runtime. The system executes an activity only if the conditions of some or all of its incoming control connectors evaluate to true. The workflow designer can select whether all (AND behavior) or some (OR behavior) of the conditions must be true.

Data flow between activities is specified using data connectors that link pairs of data containers. Before starting an activity, the system fetches the activity's input data items by following its incom-

ing data connectors. Data flow is allowed only between activities that are linked (directly or indirectly) by control connectors. Larger processes can be structured using one of two nesting constructs. *Blocks* are subunits defined only within the scope of a given process and are used for modular design and as specialized language constructs (for, do-until, while, fork). *Subprocesses* are processes used as components of other processes. Like blocks, they enable the hierarchical structuring of complex process structures. The difference is that subprocesses are included by reference, allowing a given subprocess to be part of an arbitrary number of processes. Late binding (the referenced process is read only when the subprocess is started) allows dynamic modification of a running process by changing its subprocesses. Blocks and subprocesses enable the structured modeling of workflows, which leads to a process structure resembling a tree of tasks, where the intermediate nodes are blocks and subprocesses and the leaf nodes are always activities.

ARCHITECTURE

We can divide a WFMS into three

functional areas (see Figure 3): Buildtime, Runtime Control, and Runtime Interactions. The Buildtime functions support the definition and modeling of workflow processes. The Runtime Control functions handle a process's execution, and the Runtime Interactions provide interfaces with users and applications (see Figure 4). Of these, Buildtime and Runtime Control are likely to be centralized—the former because it will be accessible only to a small set of workflow designers, the latter because it is common to all users and usually has high demands in terms of storage capacity.

Runtime Control has two aspects: persistent storage and process navigation. Persistent storage lets the system recover from failures without losing data and also provides the means to maintain an audit trail of process execution. The navigational logic controls the execution of processes by changing the status of activities according to the evaluation of conditions. Therefore, Runtime Control has two basic components, the storage server and the navigation server. The Reference Model refers to these as the Workflow Control Data and the WFM Engine. Similarly, Runtime Interactions are of two types: interactions with users and interactions with invoked applications. The former is the interface with end users and consists mainly of the work list assigned to a given user. The latter is the interface to the applications being executed as part of a workflow. These are usually considered separate components, the User Interface and the Application Interface. These appear in the Reference

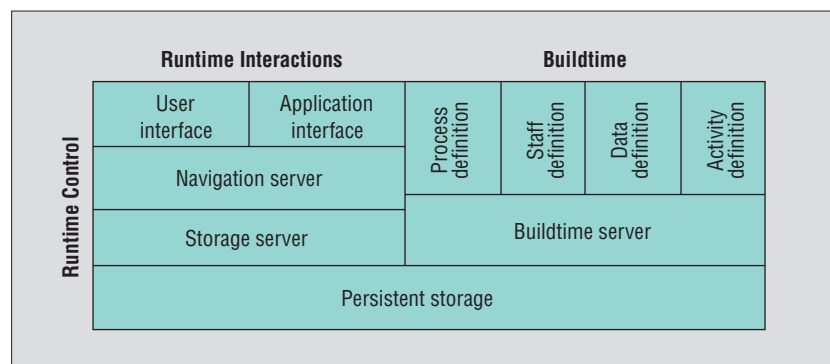


Figure 3. Overall architecture of a workflow system.

Model as Worklist and Invoked Applications. Note that, in practice, a user always owns a work list. To access the list, a user logs in to a PC or workstation and uses the user interface provided. Then, an application interface launches in that PC or workstation to drive the execution of local applications. Applications that run on machines where users do not usually log in, such as mainframes, are driven by an application interface specifically designed for those machines.

The Buildtime component is the development environment. It provides a language (usually through a graphical interface) for defining processes as well as tools for debugging and compiling the source process definition into an executable object process definition. In most systems, this implies translation to some form of database schema. It also provides tools for defining the staff associated with the system (users, roles, management levels, groups, and authorizations associated with these entities), the tools associated with the activities (programs to invoke, network addresses of computers where they reside, parameters to be passed, access paths, and so on), and the data flow between activities (usually through a mapping between data containers defined within the process).

These components can reside in different nodes and run on different operating systems. To support such diversity, WFMSs incorporate a communication

layer that hides the underlying communication protocols. This layer provides a set of standard primitives for the different workflow entities to communicate with each other, thus facilitating component portability. The communication layer performs additional functions such as acting as a name server, providing reliable communication, and even performing primitive forms of load balancing.

LIMITATIONS OF EXISTING SYSTEMS

So far, the functionality provided in commercial systems has determined the state of the art in workflow management. Paradoxically, this has been the major source of limitation. Many products were developed without a clear understanding of the user requirements and, as any serious workflow practitioner can testify, these products were quite unprepared to meet the demands that eager users placed on them.

To understand this, you must understand the background of workflow management. We can trace the direct ancestors of commercial workflow systems back to work done in office automation, image processing, and computer-supported cooperative work. In these environments, the main problems were sharing and cooperation; unfortunately, the developers rarely considered issues such as scalability or reliability. Few of the early commercial products were based on online transaction

processing (OLTP) or database technology. Although many of them used databases as the underlying repository and some incorporated functionality-related ideas, workflow systems were not conceived to deal with the daunting tasks that very large databases or sophisticated transaction processing monitor installations face. As a consequence, the robustness and technological maturity reached in these areas is all but lacking in workflow systems.

We attribute other restrictions of current technology to the centralized architecture defined by the Reference Model.³ This architecture is appropriate as long as business processes execute within the bounds of an enterprise. If the processes to be automated cross organizational and geographical boundaries and need to integrate many participants over the Internet, new decentralized architectures for WFMSs are required to avoid bottlenecks.⁶ Distributed WFMSs are also the key to increasing the availability and scalability of the workflow service.

Fault-tolerant workflow systems

Of the many aspects of fault tolerance, we will concentrate on two: exception handling and replication strategies for increased availability.

INCREASING AVAILABILITY USING REPLICATION

One of the major limitations of existing systems is their minimal resilience to failures. Current products have a single point of failure (the database) and no mechanism for backup or efficient recovery. This is not as much a flaw as a design decision: these products were initially intended for small groups and small loads. Large WFMSs will involve several thousand users, hundreds of thousands of concurrently running processes, and several thousand sites distributed over wide-area networks. They will be critical systems, and as such, their continuous availability will be crucial, in the same way that continuous availability is the key to many banking and corporate database applications. Companies cannot rely on a WFMS if a single database fail-

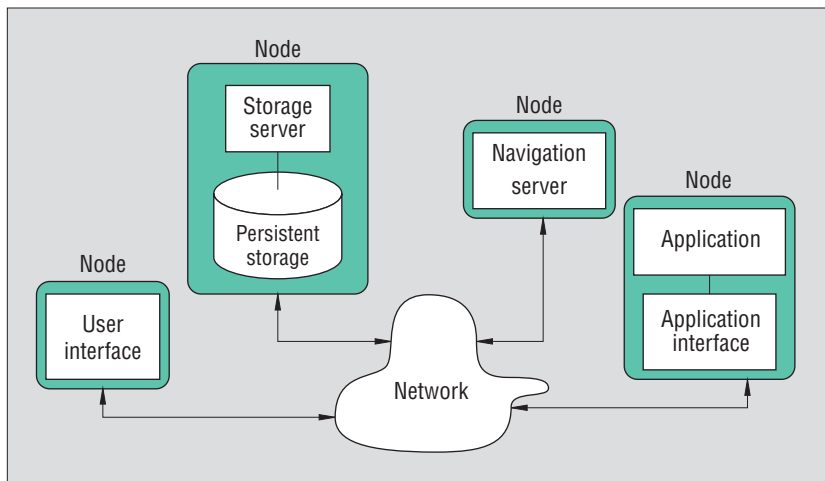


Figure 4. The runtime architecture of a workflow system.

ure can bring the entire system to a halt. Moreover, because workflow systems will operate in large distributed and heterogeneous environments, a variety of components will be involved in the execution of a process. Any component can fail, and we can do little about it today. Most systems lack the redundancy and flexibility necessary to replace failed components without interrupting ongoing work. Even if several databases are used to minimize the impact of failures (by running different processes off different databases), existing designs will stop executing all the processes associated with the failed database.

However, the availability problem has been solved in databases using different techniques. Because WFMSs are built on top of databases, it should be possible to apply these techniques to the underlying database to provide higher availability. The most common technique to do this is replication, with a mirror system synchronized with the main system. When the main system fails, the mirror takes over. If the mirror is an exact replica of the main system (all updates to the main system are also performed at the mirror), the technique is known as *hot standby*. This usually requires a Two-Phase Commit (2PC) protocol between the main and the mirror, but it lets the mirror take over almost immediately in the event of a failure. Letting the mirror stay slightly out of date instead of completely synchronized can reduce the cost. The mirror can also provide *cold standby* support by just storing the updates, without applying them, until it must actually take over.

There are, however, some differences between databases and workflow environments. First, databases assume that the primary and the backup are the same database. This would tie a WFMS to the platforms where the database runs. Second, database backups are managed at a low level (pages or log records, for instance) and replication takes place regardless of the application semantics. In a WFMS, it is possible to use the application semantics to reduce replication overhead by maintaining copies

only of events relevant to the overall execution.

To address these issues, current proposals have suggested having no dedicated backup and letting different processes have different guarantees.⁷ The reason not to have a dedicated backup is that the architecture's distributed and heterogeneous characteristics would require either a backup for every individual system or a single remote backup for the entire system, which is distributed over a WAN. Such an approach would incur too high a cost and would need to cope with the heterogeneity of

Providing different availability levels gives workflow administrators the opportunity to trade runtime overhead for recovery time.

the primary databases. Instead, databases are used both as primaries and backups: for some servers the database acts as the primary, and for others it acts as a backup.

Partly to reduce the overhead at the backup and partly to accommodate the many different requirements of workflow applications, workflow designers organize processes according to three categories. *Critical* processes are those for which execution must be immediately resumed in case of failures. Hence, they are replicated using a hot standby approach; the system forwards all changes performed at the primary to the secondary, where they are immediately applied. Both transactions, at the primary and the backup, are committed using 2PC. *Important* processes are those which should be eventually resumed in the event of failure, but some delay is acceptable. This minimizes the impact on performance because the backup does not update anything; it simply stores the changes in case they are needed to

restore the process state. When a failure occurs, the system must apply all the stored changes at the backup before resuming execution. Finally, *normal* processes rely only on forward recovery to deal with failures. They are not replicated at all, and the only guarantee is that, in case of failure, the system will resume execution where it stopped once the failure is repaired. The workflow designer assigns each process to one of the categories.

Interestingly, this backup schema is based on the application semantics, so it can be performed over heterogeneous databases. Heterogeneous database environments use a data-mapping mechanism so information from one database can be used in another database. This data mapping uses a canonical representation based on the workflow specification, so interdatabase communication takes place at the level of workflow concepts (activities, processes, data containers, control connectors, and so on). We use this same representation to avoid the problem of having to deal with internal database representations. The backup schema replicates the state of workflow entities (for instance, "activity *x* has started") rather than low-level items such as objects, tuples, attributes, or pages. Because the number of entities is relatively small, the mapping is not complicated and does not add significant overhead.

The performance penalty paid to achieve high availability is comparatively small. Figure 5a shows the results of a performance study evaluating the runtime overhead of various replication strategies.⁸ The graphic shows the average duration for a typical navigation transaction depending on the replication protocol used and the number of concurrent processes at each server. A navigational transaction at the workflow server encapsulates the time between the end of one activity and the start of the next according to the process description (this is a reasonable unit of execution because it is independent of process size, which might vary considerably). The transaction comprises updating multiple objects such as data containers, states of control connectors, and the affected

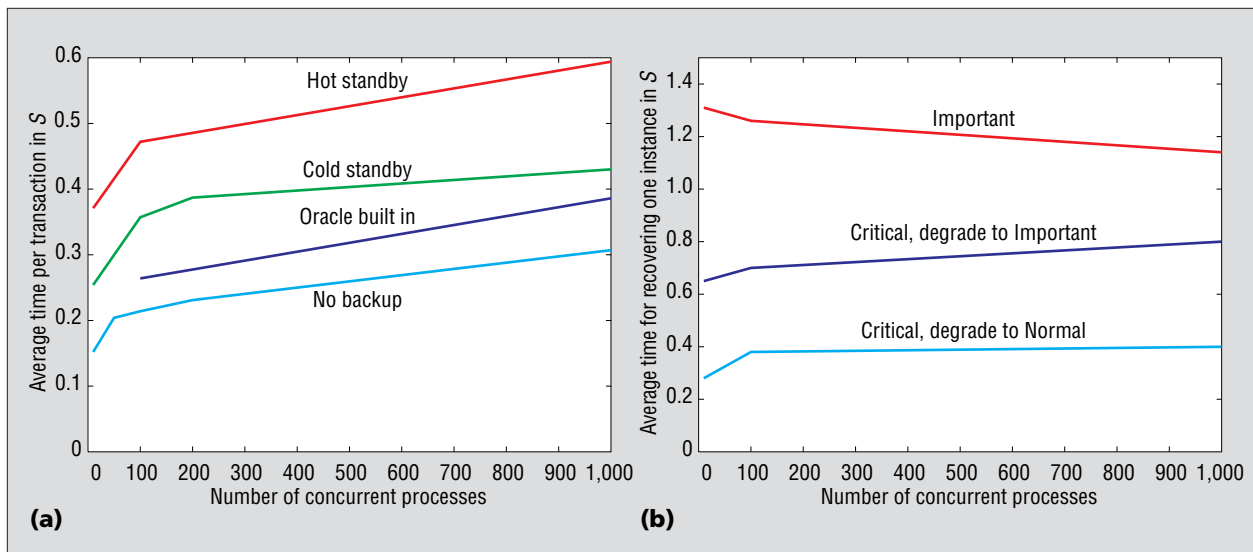


Figure 5. (a) Runtime overhead and (b) recovery cost for various replication strategies.

process's state as well as analyzing the process description to determine the next programs to start, thus incorporating the update of several schema objects. The number of concurrent processes on the horizontal axis indicates the database size and has a significant impact on the performance of all operations.

The most efficient replication strategy, using the replication services of a commercial database system (our study used Oracle8), incurred a runtime overhead of only 30% for hot standby replication. Although this indicates the high degree of optimization in current database technology, it is of little practical use for implementing workflow backup, because it ties the workflow system to a specific database architecture. Fortunately, the runtime overhead of implementing the replication at the application level instead of at the database level is still acceptable, rising to at most 90% in the worst case, which is for critical processes and a large database (1,000 concurrent processes). The actual time required is approximately 0.6 seconds. This time lapse is easily masked in many applications because other operations such as communicating with clients, updating graphical user interfaces, transporting data between machines, and starting an application are quite expensive and are likely to be the real source of delays.

The strategy of providing different availability levels gives the workflow administrator the opportunity to trade runtime overhead for recovery time, as Figure 5b shows. The recovery time per process is significantly lower for critical

processes, especially if they can be "demoted" from *critical* to *important* availability during the recovery procedure. The reason is that, without the demotion, the system must ensure that each process is still installed in two databases, which requires copying the whole process to a new backup database. With the demotion option, only the change information must be stored at the backup database. Assuming that two consecutive server failures rarely affect the same process, this strategy seems the best option.

EXCEPTION HANDLING FOR INCREASED FAULT TOLERANCE

Another significant drawback of existing workflow systems is their poor support for exception handling. This is a major obstacle for large-scale and mission-critical applications. Most workflow applications are confronted with exceptions such as failures of invoked applications, communication failures, lack of response from a user, missed deadlines, and unexpected behavior of applications. The usual failure-handling procedure in most systems is to stop process execution and report the failure to an administrator. However, as workflow applications become larger and more complex, manual failure resolution becomes less and less feasible because of the demand for human resources, with their high cost and slow answer time. Clearly, we need automatic exception handling, especially for scalable systems.

The problem of exceptions and fault tolerance is not particular to WFMSs; indeed, the problem occurs in almost all

complex software systems. It should, in principle, be possible to apply concepts already developed in systems such as database management systems, programming languages, operating systems, or distributed computing platforms. The first two areas just listed are particularly promising, and a number of research projects have recently investigated their application to workflow environments. Adapting known techniques to workflow management is the main challenge here.⁹ For instance, database transactions, which comprise a powerful abstraction for failure handling, are inefficient when used in applications other than OLTP. For workflow environments, where processes tend to be long-running and resources are usually outside the scope of the WFMS, we need modified notions of a transaction that take this into account. Likewise, we must also reconsider the semantics of traditional exception-handling concepts for programming languages if we are to use them in workflow systems. A recent approach aims at combining database transaction concepts and programming-language ideas into a coherent paradigm for fault-tolerant workflows.¹⁰ The concept is based on enhancing the workflow model with new concepts such as atomic spheres, exception signals, and exception handlers.

Atomic spheres are sets of activities that must either execute completely or not at all. Introducing them into the workflow modeling language facilitates the wrapping of activities, blocks, or subworkflows into such spheres. This gives the workflow designer the means to model

processes as nestings of spheres, each of which can be separately undone if an error occurs during its execution, thus effectively limiting a failure's impact to the scope of that sphere. This concept bears a strong resemblance to advanced transaction models, which were originally designed to overcome the shortcomings of the traditional transaction concept.

One improvement has been the modification of the traditional notion of atomicity from all-or-nothing to a more relaxed definition, known as *semiatomicity*. This new correctness criterion lets us define a sphere as successfully executing even if some of its activities fail, as long as the system can find successful alternative executions and reach pre-determined final states. From a formal point of view, this lets us undo part of the work, execute contingency activities, and still treat the result as a correct execution. Even though atomic spheres and semi-atomicity enable fault-tolerant workflows, the problem of modeling failure handling in an effective and comprehensible way remains. Here, programming language concepts are useful.

The concept of exception signals and exception handlers came from the many newer programming languages (Ada, ML, C++, Java), operating systems (Windows NT), and distributed execution platforms (CORBA). Its aim is to make the workflow model more robust and comprehensible by separating "normal" control flow from the failure-handling logic. To accommodate this notion into workflow processes, we embed all failure-handling logic in exception handlers attached to atomic spheres. An exception handler can be an arbitrary flow, so we can implement any failure-handling scheme necessary. When detecting an exception in a sphere, the WFMS passes control to the appropriate handler, which executes the tasks necessary to repair the failure. Next, depending on the type of handler, it will either abort the sphere (thus effectively implementing a contingency plan) or resume execution. A third possibility is propagating the exception to the enclosing sphere; this might ultimately produce an abort of the whole process if, for example, a failure occurred

for which no appropriate handling was specified. One of the challenges of using this approach for workflow languages is its integration into graphical languages, which are usually used for workflow modeling. To accommodate convenient workflow modeling, language primitives must be seamlessly integrated into whatever modeling flavor the workflow designers know.¹⁰

Exception signals and exception handlers let workflow designers describe a process's failure semantics in a convenient way. Additional mechanisms are necessary to enforce these failure semantics: for instance, "aborting a task" must be properly defined. In practice, aborting a task and executing an exception handler is equivalent to what is known as *partial backward recovery* in advanced transaction models. An important issue in workflow contexts is how to deal with application semantics if spheres must be rolled back. Many programs are not inherently atomic, an assumption that advanced transaction models generally make.

Also, only some of the activities that a workflow executes will support physical undo as, for example, database management systems implement it. Instead, the vast majority of applications must use compensation, logically undoing the effects of activities. This again requires extensions to modeling languages, because the workflow programmers need to specify undo methods for activities that the WFMS can then use if a sphere must be undone. In practice, the range of possible undo methods can be wide, and the WFMS must support all of them. For instance, in the simplest case, an undo method will be the execution of a single program that is logically inverse to the one that must be compensated; in other cases, compensation may require the execution of a whole subworkflow. In yet other cases, a compensating activity for a whole workflow might exist that can speed up recovery significantly, because not every single activity must be compensated. Hence, recovery mechanisms for workflow systems must be much more pragmatic and flexible than the known mechanisms in workflow systems. Because of this, many researchers see workflows as the first real

application area for advanced transaction models, which have until now mainly remained theoretical concepts.

The concept of exception signals and exception handlers can handle all kinds of exceptions flexibly, as long as the workflow designers have anticipated them. How to handle unexpected exceptions—that is, failures that the workflow designer did not foresee—is still an open research topic. The occurrence of such an exception usually indicates an erroneous or incomplete workflow specification, which therefore requires modifications to the process. However, the WFMS must gracefully handle even this situation to avoid losing the work performed before the error occurred. As in the case of expected exceptions, it is usually not acceptable to abort the whole workflow. Instead, to capture the new situation and resolve it coherently, we need mechanisms to dynamically modify running workflow instances. Dynamic workflow modification is also an open research topic,¹¹ but because of its importance, it will likely become an important facility in future WFMS generations.

CURRENT COMMERCIAL workflow management systems are inflexible, do not scale well, and do not handle failures appropriately. Ongoing research is addressing many of these issues using both known techniques from the transaction-processing world as well as innovative ideas based on new technologies. Success in incorporating these ideas into existing products will be a key factor in establishing WFMSs as one of the crucial components of corporate computing. //

References

1. G. Alonso et al., "Distributed Processing over Stand-Alone Systems and Applications," *23rd Int'l Conf. Very Large Databases (VLDB '97)*, Morgan Kaufmann, San Francisco, 1997, pp. 575–579.
2. G. Alonso and C. Mohan, "WFMS: The Next Generation of Distributed Processing Tools," *Advanced Transaction Models and*

Coming Next Issue

Applied Operating System Research and Development

Guest Editor:

Dejan Milojevic
Hewlett Packard Laboratories

Even though there are fewer general-purpose operating systems developed today, research scientists and technology developers have not lost interest in the field. The OS field has somewhat refocused in the past few years, moving toward Web technology and middleware. Many OS techniques are now being deployed in new settings, such as on the Internet and in various programming environments (Java, Tcl/Tk, and so forth). This special issue will feature applied OS research and development.



- Architectures*, S. Jajodia and L. Kerschberg, eds., Kluwer Academic, Hingham, Mass., 1997, pp. 35–62.
3. D. Hollinsworth, *The Workflow Reference Model*, Workflow Management Coalition, Tech. Report TC00-1003, Dec. 1994; www.wfmc.org (current July 2000).
 4. J. Bacon, ed., *Special Issue on Workflow Management Systems*, *IEEE Concurrency*, July–Sept. 1999.
 5. D. Georgakopoulos, M. Hornick, and A. Sheth, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure," *Distributed and Parallel Databases*, Vol. 3, No. 2, Apr. 1995, pp. 119–153.
 6. S. Paul, E. Park, and J. Chaar, "RainMan: A Workflow System for the Internet," *Usenix Symp. on Internet Technologies and Systems*, Usenix, Berkeley, Calif., 1997.
 7. M. Kamath et al., "Providing High Availability in Very Large Workflow Management Systems," *Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96)*, Springer Verlag, Berlin, 1996, pp. 427–442.
 8. C. Hagen and G. Alonso, *Backup and Process Migration Mechanisms in Process Support Systems*, Tech. Report No. 304, ETH Zürich, 1998.
 9. G. Alonso et al., "Advanced Transaction Models in Workflow Contexts," *Proc. 12th Int'l Conf. Data Engineering*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 574–581.
 10. C. Hagen and G. Alonso, "Flexible Exception Handling in the OPERA Process Support System," *18th Int'l Conf. Distributed Computing Systems (ICDCS)*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1998, pp. 526–533.
 11. M. Klein, C. Dellarocas, and A. Bernstein, "Towards Adaptive Workflow Systems," *Proc. CSCW-98 Workshop*, 1998; <http://ccs.mit.edu/klein/cscw98> (current July 2000).

Gustavo Alonso is an assistant professor in the Department of Computer Science at the Swiss Federal Institute of Technology in Zürich, where he leads the Information and Communication Systems Research Group. His research interests include cluster computing, databases, workflow management, scientific applications of database and cluster technology, transaction management, as well as replication, availability, and scalability in databases and clusters. Previously, he was a visiting scientist at IBM Almaden Research Center, where he participated in the Exotica project working on advanced transaction models and workflow management systems. He received a degree in telecommunication engineering from Madrid Technical University and an MS and PhD in computer science from the University of California, Santa Bar-

bara. Contact him at the Inst. of Information Systems, ETH Zentrum, CH-8092 Zürich, Switzerland; alonso@inf.ethz.ch.

Claus Hagen is an information systems architect at Credit Suisse, a major Swiss bank, where he is responsible for workflow management and middleware architectures. He received his diploma degree in computer science from the University Erlangen-Nürnberg and his PhD in computer science from the Swiss Federal Institute of Technology. Contact him at Credit Suisse, Dept. CIXT, CH-8070 Zürich, Switzerland; claus.hagen@credit-suisse.ch.

Divyakant Agrawal is a professor of computer science at the University of California, Santa Barbara. His research interests include distributed systems, computer networks, databases, and large-scale information systems and digital libraries. He received his BE from Birla Institute of Technology and Science, Pilani, India, and his MS and PhD in computer science from the State University of New York at Stony Brook. Agrawal is a member of the ACM and the IEEE Computer Society. Contact him at Dept. of Computer Science, Univ. of California, Santa Barbara, CA 93106; agrawal@cs.ucsb.edu.

Amr El Abbadi is a professor of computer science at the University of California, Santa Barbara. His research focuses on fault-tolerant distributed systems, databases, and information systems. He received his PhD in computer science from Cornell University and has been a visiting scientist at IBM Almaden Research center as part of the Exotica project. He is the area editor for *Information Systems: An International Journal*. Contact him at the Dept. of Computer Science, Univ. of California, Santa Barbara, CA 93106; amr@cs.ucsb.edu.

C. Mohan is an IBM Fellow at the IBM Almaden Research Center. His research focuses on concurrency control, recovery, transactions, storage management, indexing, query processing, and workflow management. After receiving his PhD in computer science from the University of Texas at Austin, he joined IBM where he founded and led the Exotica and Dominotes projects. His results have been incorporated in DB2, MQSeries, and Encina and in the X/Open XA standard. He received the ACM SIGMOD Innovation Award in 1996 for inventing the Aries family of locking and recovery protocols and the industry-standard Presumed Abort commit protocol. Contact him at IBM Almaden Research Center, 650 Harry Rd., K55/B1, San Jose, CA 95120; mohan@almaden.ibm.com; www.almaden.ibm.com/cs/people/mohan.